AL-TP-1991-0050

A THEORY OF AUTOMATED DESIGN OF VISUAL INFORMATION PRESENTATIONS

Stephen Westfold
Cordell Green

Kestrel Institute
3260 Hillview Avenue
Palo Alto, CA 94304

for

Systems Exploration, Incorporated
5200 Springfield Pike, Suite 312
Dayton, OH 45431

ARMSTRONG LABORATORY

HUMAN RESOURCES DIRECTORATE
LOGISTICS RESEARCH DIVISION
Wright-Patterson Air Force Base, OH 45433-6503

February 1992

Interim Technical Paper for Period June 1990 – May 1991

AIR FORCE SYSTEMS COMMAND
BROOKS AIR FORCE BASE, TEXAS 78235-5000

NOTICES

This technical paper is published as received and has not been edited by the technical editing staff of the Armstrong Laboratory.

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely Government-related procurement, the United States Government incurs no responsibility or any obligation whatsoever. The fact that the Government may have formulated or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication, or otherwise in any manner construed, as licensing the holder, or any other person or corporation; or as conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

The Office of Public Affairs has reviewed this paper, and it is releasable to the National Technical Information Service, where it will be available to the general public, including foreign nationals.

This paper has been reviewed and is approved for publication.

DAVID R. GUNNING
Task Monitor

BERTRAM W. CREAM, Chief
Logistics Research Division

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	February 1992	Interim Paper – June 1990 – May 1991

4. TITLE AND SUBTITLE
A Theory of Automated Design of Visual Information Presentations

5. FUNDING NUMBERS
C – F33615-88-C-0004
PE – 63106F
PR – 2950
TA – 00
WU – 21

6. AUTHOR(S)
Stephen Westfold
Cordell Green

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Kestrel Institute
3260 Hillview Avenue
Palo Alto, CA 94304

Systems Exploration, Incorporated
5200 Springfield Pike, Suite 312
Dayton, OH 45431

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)
Armstrong Laboratory
Human Resources Directorate
Logistics Research Division
Wright-Patterson Air Force Base, OH 45433-6503

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AL-TP-1991-0050

11. SUPPLEMENTARY NOTES
Armstrong Laboratory Technical Monitor: David R. Gunning, (513) 255-2606

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This paper reports on an elaborated theory of graphical display that describes how to automate the generation of tables and diagrams to represent relations occurring in technical data. This theory is an extension and reformulation of our earlier theory. It includes a semantic framework for expressing the translation of relations into graphics. The framework allows for fine-grained control of synthesis and for human factors guidance. Methods are presented for synthesis of multi-page and multi-screen graphics, for the graceful integration of tables and diagrams, for composite maps such as used for color coding, and for development of plot charts.

14. SUBJECT TERMS
Automated generation of formats
Automated technical data
Computer generated maintenance aids

15. NUMBER OF PAGES
40

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102

TABLE OF CONTENTS

Page

I. INTRODUCTION 1

II. OVERVIEW OF THEORY 3

III. LANGUAGES 4

Data Language 5
Visual Language 5
Transformation Language 7

IV. BASIC VISUAL REPRESENTATIONS 9

Adjacency 10
Arcs 10
Containment 12
Visual Attributes 12
Interference and Ambiguity 13

V. DERIVATIONS 14

Composite Sequences 14
Table Derivations 14
Diagrams and Trees 19
Map Decomposition 23

VI. CONCLUSIONS 27

REFERENCES 29

LIST OF FIGURES

Figure

1 Example of the Three Stages of Visual Presentation 3
2 Rendering of **join(*a*, *1*)** Representing **<"a", "1">** 11
3 Example of a Call-out 11
4 Rendering of **contain({*a*, *b*})** Representing **{"a", "b"}** 12
5 Sample Set Layout 23

PREFACE

The work described in this technical paper was performed at the Kestrel Institute under subcontract from Systems Exploration, Inc. The work was accomplished for the Air Force Human Resources Laboratory under Contract Number F33615-88-C-0004, Task Order Number 20. The work is in support of AFHRL Project 2950, Integrated Maintenance Information System. Mr. David R. Gunning monitored the contract for AFHRL.

This paper elaborates on our theory for the automated display of neutral maintenance data. The feasibility of this technology was described in our earlier report, "Automated Design of Displays for Technical Data," AFHRL-TP-90-66. Mr. Gunning suggested that the theory could be elaborated and refined to allow graphics to scale up to multi-screen images, thereby allowing the graceful integration of tables and diagrams. This report presents the new, elaborated theory.

We would like to acknowledge David Gunning, Peter Ladkin, David Zimmerman, and Richard Jüllig for their assistance and suggestions.

SUMMARY

This paper presents an elaborated and reformulated theory documented in an earlier report, "Automated Design of Displays for Technical Data," AFHRL-TP-90-66, which describes automated design rules for display of technical data. The theory can be employed to create displays of tables, diagrams, trees, and hybrid mixtures thereof.

The new capabilities presented herein include finer-grained control of synthesis; closer, seamless integration of tables and diagrams (including tables with arcs); composite maps to represent color coding, off-page pointers, or other forms of indirection; containment trees; explication of plot chart derivations; and multi-screen graphics.

A semantic framework for expressing the theory of synthesizing and rendering graphic representations of data is also presented. The framework includes formal languages for describing the data, visual attributes, and transformations.

The concept of operations for the Integrated Maintenance Information System requires that all technical data be stored in a data base, independent of screen formatting information. For example, the electronic form includes no information about where a box of a diagram is to be placed, or even that some information is to be displayed as a diagram instead of a table. Hence, this *format-neutral* storage representation allows software to automatically reformat the data for new computer screen sizes, portables, color capabilities, or methods including animation or head-mounted displays. This separation of information content from formatting also permits better analysis to check for consistency or to merge information from multiple sources. When there are changes, minimal human re-engineering effort is required to update the information and the displays, resulting in reduced costs and higher quality.

I. INTRODUCTION

This paper presents a theory for the visual display of electronic relational data. It assumes that data are stored in format-free form (using high-level data structures such as sets and sequences). The mappings and transformations defined herein are used to add desired structure during display so information is presented in the most appropriate form.

A method such as ours is essential in an environment in which new information is constantly being introduced or multiple versions of data must be continuously maintained. For example, introducing a new connecting cable between electronic devices requires modifying the maintenance manual in a non-local manner. Not only is a new entry required for the connecting cable, but entries for the parts connected by the cable must be modified as well. Further, test procedures, fault trees, and decision tables are all affected by the introduction of the new part. If the data are kept in a format-free, object-oriented form such as the Content Data Model (CDM), modifications to the data model are relatively easy and reformatting is an automatic by-product of our display methodology. In contrast, reformatting the manuals by hand is time-consuming and error-prone.

Furthermore, our methodology enables one to build a formal model of the data integration, update, and reformatting process; this model may then be formally verified by computer. Verification proves that implementing the integration and formatting in software is consistent with the data model. Verification is infeasible with manual data update procedures. The formal model also allows one to prove other properties of the display process and its implementation in software (e.g., that warnings for any maintenance procedure must accompany a screen display of that procedure or that every action in a vendor-supplied list of diagnosis actions is displayed during a diagnosis procedure for the subsystem).

The theory presented here underlies and extends results presented in Westfold et al. (1990). This theory, based on successive transformations of data from stored form to display form, includes visual properties of displays such as color coding and containment. It also focuses on the generation and display of tables and diagrams, and describes in detail the transformations required for these presentation types and their meanings. A formal model of the data update, enhancement, and display process allows for the use of methods, techniques, and tools similar to those used in the formal development of software systems.

Our formulation of the problem is essentially that of MacKinlay (1986); however, our analysis aims for finer granularity and a broader scope so that more types of visual representation can be synthesized. To describe a general theory of visual representation, a *data language* and a *visual language* is defined to describe the source and target domains, respectively, format-free data and visual displays. The display of data in a particular format is divided into three stages: transforming data language descriptions into (a) other data language descriptions, (b) visual language descriptions, and (c) displays (*rendering*). *Representation* is the composition of these three transformations. Representation is a one-to-many relation; data can be represented in many ways, but each visual display of information represents a unique data set (otherwise the display would be *ambiguous*). *Interpretation* is the inverse of representation; it is a many-to-one function. These stages are detailed in Section II.

The data language includes data constructors such as strings, sequences, sets, and finite maps. Visual representations include adjacency and arcs for pairs of regions; containment for sets of regions; text boxes for strings; visual attributes (such as color, shape, font, and position) for maps; and lengths for quantities. The language used to express the transformation rules is standard first-order predicate logic enhanced by operators for manipulating sets, relations, and maps. These languages are described in Section III. Section IV formally describes the basic visual representations of adjacency, arcs, and containment. It also briefly considers the properties of interference and ambiguity which may arise when the functionality condition on interpretation is not maintained.

Applying specific transformation rules to examples is called *derivation*. Section V formally specifies the derivations of tables (including tables with and without labeled columns) in one-page or multi-page form, with distributed labels for each item, and interconnected by arcs within the table. The derivation of a block-diagram representation of system-subsystem connectivity for a radar control system is also specified. Section V shows how map decompositions are used to communicate data relations between properties and derive plot charts. Finally, our research conclusions are stated in Section VI.

II. OVERVIEW OF THEORY

Figure 1 illustrates the three stages of deriving a particular tabular display for the simple relation {<"a","1">, <"b","1">, <"b","2">}.

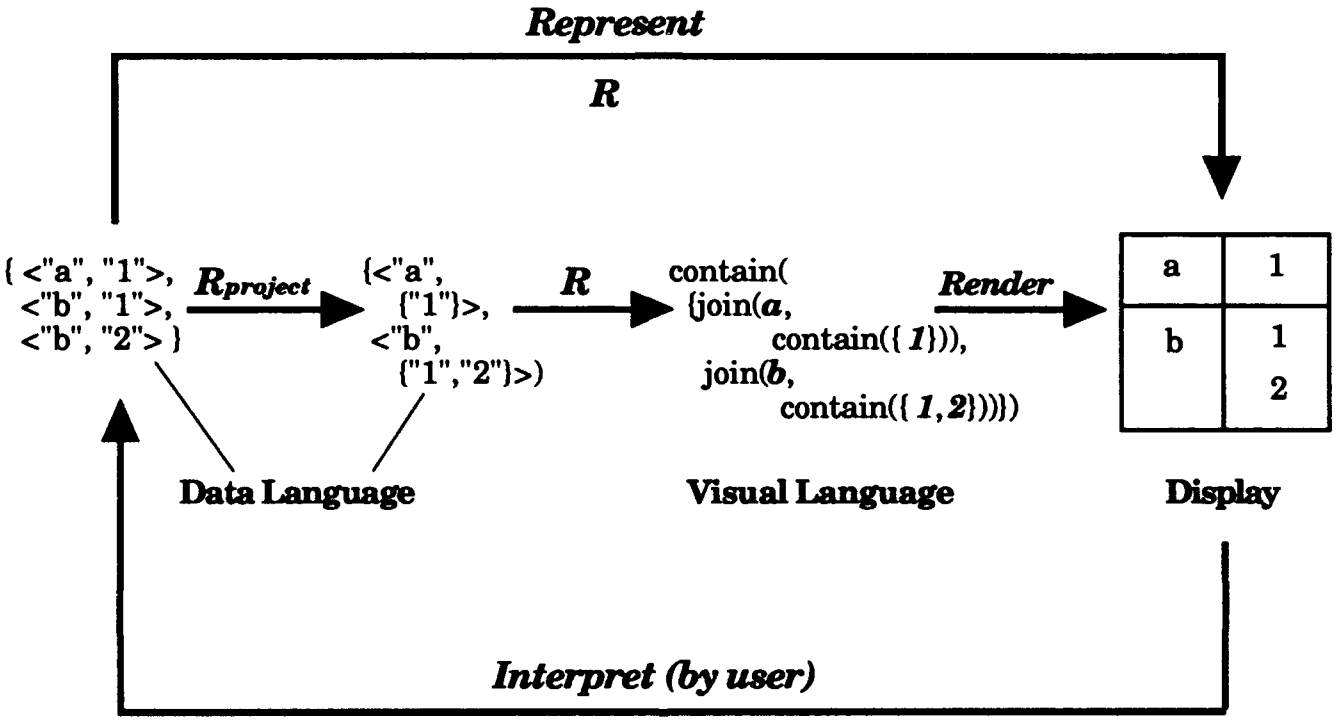


Figure 1. Example of the Three Stages of Visual Representation.

Stages 1 and 2 are broken into smaller transformations. This paper is not much concerned with Stage 3, assuming for the most part a straightforward rendering of visual language expressions with some simple constraint solving. **R** is the representation relation, relating specifications to presentations. **I** is the interpretation function.

In general, a derivation of a visual display consists of a sequence of zero or more transformations within the data language, followed by a transformation into the visual language, and, finally, a rendering of the resulting visual language statement. Each transformation specifies both how to convert the source expressions into target expressions and how to interpret target expressions as source expressions. Each individual transformation is specified as a relation between its input and output; a relation which, like **R**, is one-to-many in general, and whose inverse, therefore, is an interpretation function. The interpretation of the visual expression consists

of applying the individual interpretations for each transformation in the reverse order. A visual expression is ambiguous if it is the result of two different transformation sequences with different starting specifications.

Our transformations preserve information so that the interpretation function is defined, but they do not preserve equivalence. Thus, for the representations to work, the user must have some knowledge of these transformations; the transformations may transform between forms the user perceives as essentially the same, or they may embody standard conventions such as those for line and page breaking.

In general, many visual representations exist for a given set of data language statements; therefore, it is desirable to have a method that exist allows comparison of different representations. A cost function on interpretation would provide such a method, as a primary measure of a representation must be how easy it is to interpret. If a cost function is available to interpret each transformation, these costs could be summed to provide a total cost for the whole interpretation. This paper largely ignores the cost of interpretation; rather, it concentrates on specifying when a unique interpretation is possible and showing the derivations of a large variety of displays.

The primitive visual representations are as follows:

- adjacency for pairs,
- arcs for pairs,
- containment for sets,
- text boxes for strings,
- visual attributes (color, shape, font, x-position, y-position) for maps, and
- lengths for quantities.

This list is similar to that from Bertin (1983) and MacKinlay (1986). Using these primitives, more complex structures for tables, plot charts, and box and arrow diagrams can be derived.

III. LANGUAGES

The specification of relational information consists of a set of expressions in the data language. This is transformed into a set of expressions in the visual language that represent the original expressions. This set of visual language expressions is then rendered in a display.

Data Language

The data types and constructors of the data language are as follows:

Type	Expression in Data Language
string	**"a string"**
Boolean	**true, false**
tuple or sequence	**<x, y, ...>**
set	**{x, y, ...}**
map	**{| w -> x, y -> z, ... |}**

where **w**, **x**, **y**, and **z** are expressions in this language. This simple language is sufficient to specify all the examples in this paper. Relations are modeled in this language as sets of tuples.

Although this language is sufficient to specify relational information, it may not be convenient for user purposes. A user-oriented language that would be translated to this language might be used. Such a user-oriented language may be graphical and have more structure to ensure the specification is coherent.

As an example of some of the details in a specification, consider what may be necessary to communicate a binary relation. As a minimum, the set of pairs must be shown. It may be specified that some other properties of the relation should also be shown, such as the name of the relation, the domain, the range, the name of the domain, and the name of the range. Additional constraints may be specified that require additional information to be shown, such as some well-ordering on the domain.

Visual Language

The visual language domain consists of attributed regions and values of visual attributes such as colors and shapes.[1] Regions are specified using primitive text boxes and region constructors that take one or more regions and produce a new region. All constructors have a common characteristic: they include all their region arguments as subregions, and all subregions

[1] The form of the visual language is chosen to facilitate expression of the representation relations and is not intended for general use.

within them are either arguments or subregions of arguments. Visual attributes are attached to regions by constructors that take a region and an attribute value and produce a region with that attribute value.

The primitive constants of the language are:

- ***a-string*** : the region containing the text 'a-string,'
- **red, blue, ...:** colors,
- **circle, square, ...:** shapes, and
- **<1, 27>, <9, 2>, ...:** positions.

The region constructors of the language are:

- **join(x, y):** the region containing precisely the adjacent regions **x** and **y**,
- **contain(S):** the region containing precisely the regions of the set **S**,
- **arc(x, y):** the region containing regions **x**, **y** and an arc connecting **x** to **y**,
- **color(x, y):** the region **x** with color **y**,
- **shape(x, y):** the region **x** with shape **y**, and
- **position(x, y):** the region **x** with position **y**.

For example, the expression below describes a region at position **<4, 2>** that includes an arc joining a circled red **"a"** to a square-boxed, blue **"b."**

position(arc(shape(color(*a*, red), circle),
shape(color(*b*, blue), square)),
<4, 2>)

The constructors **join** and **contain** both specify constraints on regions; their value is the region composed of the regions of their arguments. A specific definition of *adjacent*, beyond that **join** be associative, is not required.

Visual attributes (color, shape, and position) are modeled as region constructors of two arguments: the constructor value is the region of the first argument, the attribute value is given by the second. Modeling attributes using constructors allows complex visual descriptions to be stated with expressions rather than conjunctions of expressions. This, in turn, allows the representation equations below to be stated more simply.

Since **join** is associative, it is natural to extend the binary **join** to an n-ary **join**:

join(x, y, ...) = join(x, join(y, ...)).

To specify the properties of **contain**, it is convenient to introduce the region predicate **within** that corresponds to set membership. It is defined as follows:

x within y <=> ex(S) (x in S & y = contain(S)).

For example, if

y = contain({*a*, *b*})

then

***a* within y** and ***b* within y** but **~*c* within y.**

Transformation Language

In addition to the data and target languages, standard first-order predicate calculus is used to specify the transformation relationships. In particular, the following constructs are used.

&	- conjunction
or	- disjunction
<=>	- logical equivalence
=>	- logical implication
=	- equality
=def	- defined equality (for presenting definitions)
fa(x) p	- for all x, p is true
ex(x) p	- there exists an x such that p is true
{x | p}	- set comprehension, the set of x such that p is true
reduce(op, Q)[2]	- reduce the set or sequence **Q** by the associative operation **op**
domain, range	- domain and range of a relation represented as a set of pairs
image(R, x)	- the relational image of **x** with respect to **R**: **{y | R(x,y)}**

[2] **Reduce(op, <x, y, z>) = x op y op z**

X	- cross product: A X B = {<x, y> | x in A & y in B}
⊇	- subset predicate
∪	- set union

In logical statements, free variables are implicitly universally quantified.

Variables range over expressions in the specification, and each transformation replaces one or more variables by one or more new variables. Each transformation is specified as a relation defined by an equivalence relating expressions on the old variables with those on the new variables. The equivalence must interdefine the old and new variables; it may underspecify the new variables in terms of the old variables, but it must uniquely define the old variables in terms of the new variables for the interpretation to be unambiguous. For example, $\mathbf{R_{project1}}$, defined as:

$\mathbf{R_{project1}}$(S, S') =def
    <x, y> in S <=> ex(z) (<x,z > in S' & y in z)

gives a relation between the binary relations S and S' that includes S' as the projection on the first field of the relation S. A data language specification involving some expression, S, may be transformed to a new specification with S replaced by S' where $\mathbf{R_{project1}}$(S, S'). For example, given:

S = {<"a","1">, <"b","1">, <"b","2">}

all the following values of S' satisfy $\mathbf{R_{project1}}$(S, S') and, therefore, may replace S. However, for each value of S', only this value of S satisfies $\mathbf{R_{project1}}$(S, S'):

S' = {<"a", {"1"}>, <"b", {"1","2"}>}
S' = {<"a", {"1"}>, <"b", {"1"}>, <"b", {"2"}>}
S' = {<"a", {"1"}>, <"b", {"1","2"}>, <c, {}>}.

Each representation relation has an inverse that is a function--the interpretation function. The interpretation function, $\mathbf{I_{project1}}$, corresponding to $\mathbf{R_{project1}}$ that converts each S' into S is defined:

$\mathbf{I_{project1}}$(S') =def {<x, y> | ex(z) (<x,z> in S' & y in z)}.

For consistency, the equivalences with the expression involving the old variables are on the left; thus, the left-hand side will be simple. The right-hand side will often have disjunctions or existential quantification, reflecting the non-determinacy of the left-to-right transformation. For each transformation, both the definition of the representation relation and its inverse (the interpretation function whose form is often simpler) will be provided. $\mathbf{R_V}$ represents the Stage 2 representation relation that converts from the data language to the visual language; $\mathbf{I_V}$ is the corresponding interpretation function.

IV. BASIC VISUAL REPRESENTATIONS

Representations of basic objects (e.g., strings) and simple relationships (e.g., pairs, sets, and maps) can be combined to represent sequences and relations as well as more complex structures. The only representation for a string that is addressed in this paper is a text box: a region containing the string as text, possibly with font, style, and size attributes.

Two basic representations for pairs are considered: adjacency and connectedness. Connectedness consists of a line or arc between two regions. This representation leads to variations on such themes for images as the familiar "boxes and arrows" view of a system-subsystem hierarchy. There may be *additional constraints* on the positions of the regions or arcs.

Arcs can generally be used to merge the different occurrences of objects so they only have one visual incarnation. However, other constraints such as limits on the length of arcs can prevent the merging of nodes. The connecting arc between two nodes allows the nodes to be relocated arbitrarily as long as the connecting arc stretches to follow the nodes. However, human factors and other constraints limit the acceptable or correct locations of arcs and nodes. A common constraint is the need to show some ordering relation via increasing distance in some direction (an organization chart typically has lower organizational levels placed at lower y positions).

The basic visual representations together specify $\mathbf{R_V}$, the basic relation that directly relates a subset of expressions in the data language to expressions in the visual language. $\mathbf{I_V}$ is defined similarly. For each basic representation, the corresponding equations for $\mathbf{R_V}$ and $\mathbf{I_V}$ are provided.

Adjacency

The representation equations for a pair as adjacency (**join**) are as follows:

$R_V(\langle x, y\rangle, \mathbf{join}(vx, vy)) \iff R_V(x, vx) \;\&\; R_V(y, vy)$
$I_V(\mathbf{join}(vx, vy)) = \langle I_V(vx), I_V(vy)\rangle.$

These may be read, respectively, as:

> A pair may be represented visually by the join of two regions whenever the first region represents the first element of the pair and the second region represents the second element of the pair.
>
> A join of two regions is interpreted as the pair of the interpretation of the first region and the interpretation of the second region.

For example, consider the visual representation of the pair <"a","1">. If "a" and "1" are represented, respectively, by text boxes *a* and *1* (i.e., R_V ("a", a) and R_V ("1", 1)), then **join(a, 1)** is a representation of <"a","1"> (i.e., R_V (<"a","1">, **join(a, 1)**)).

Figure 2 shows a fragment of a display that renders **join(*a*, *1*)**. Note that the call-outs and dotted lines indicate regions and what they are rendering; they are not part of the display. In this display, "a" and "1" both have two representations, but only one of each is used in the representation of <"a","1">. The n-ary **join** may be used to represent n-tuples or sequences as well as pairs.

Arcs

The conditions on representing a pair as an arc are the same as those on representing a pair by adjacency (i.e., whenever two regions represent two objects, the pair of those two objects may be represented by making the two regions adjacent or by putting a line between them). Therefore, the representation equations for a pair as an arc are the same as those for a pair as adjacency with **join** replaced by **arc**:

$R_V(\langle x,y\rangle, \mathbf{arc}(vx,vy)) \iff R_V(x,vx) \;\&\; R_V(y,vy)$
$I_V(\mathbf{arc}(vx,vy)) = \langle I_V(vx), I_V(vy)\rangle.$

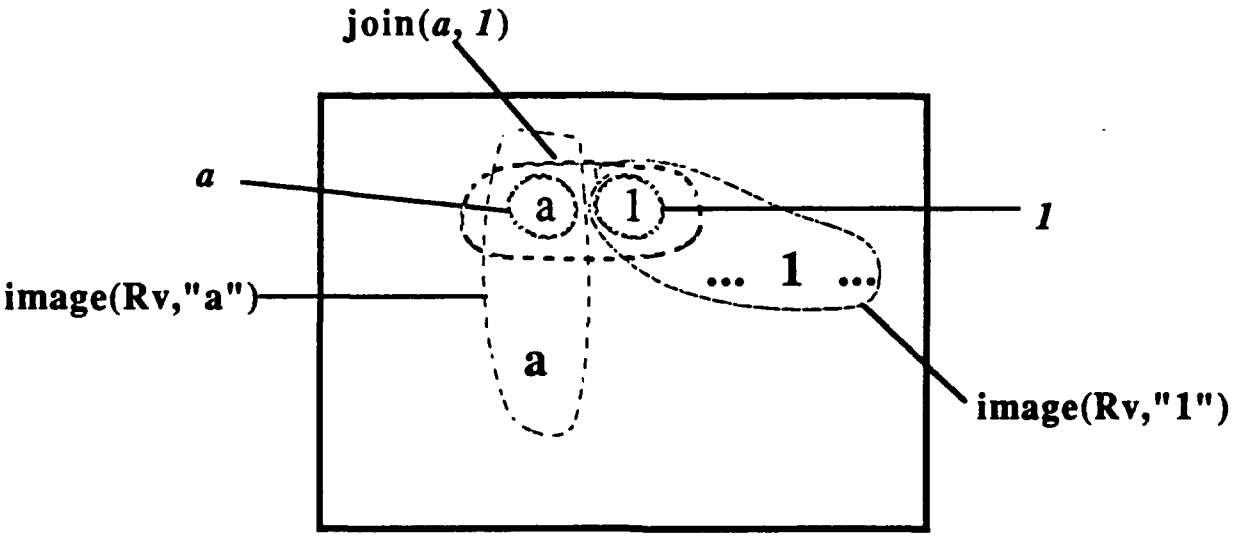


Figure 2. Rendering of **join(*a, 1*)** Representing **<"a","1">**.

A call-out is a label-pair represented as an arc. For example, consider a photograph of a subsystem containing some component that is labeled with a call-out. The label-pair representation could be **<label, component>**: the label is represented as its text box, the component is represented by a pictorial view, and the association between them is represented by a connecting line (Figure 3).

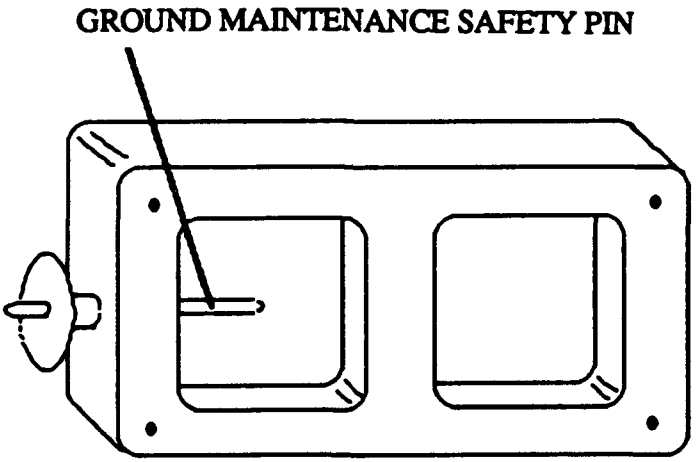


Figure 3. Example of a Call-out.

Containment

The representation equations for a set as a containing region are:

x in S <=> ex(vx, vS) (R_v (x, vx) & R_v (S, vS) & vx within vS)
I_v (contain(vSS)) = {I_v (x) | x in vSS}.

Figure 4 is an example of a set represented as a containing region. This figure illustrates the relationship between vS and vSS.

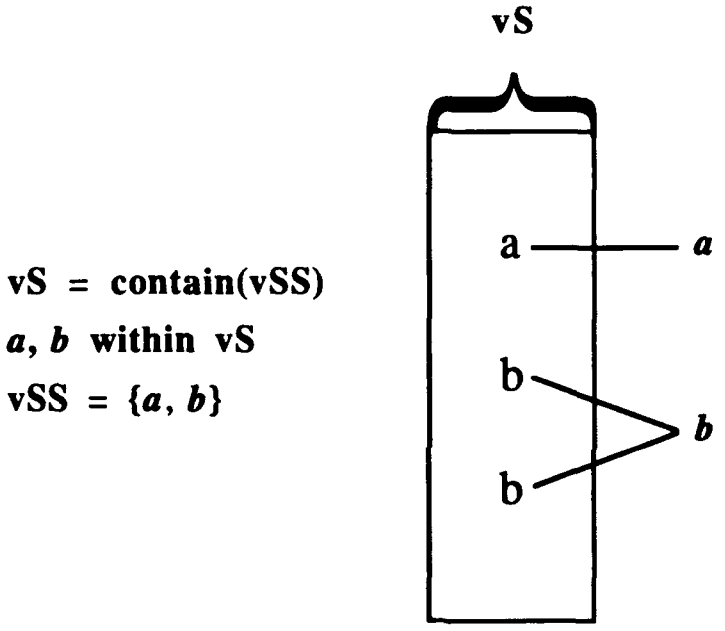


Figure 4. Rendering of **contain({*a*, *b* })** Representing **{"a","b"}**.

Visual Attributes

Visual attributes can be used to represent maps. For example, the color attribute might represent a status map: the color of a region would represent the status of the object represented by that region. The values of attributes are not regions; thus, they are very limited in what they can represent directly. The solution is to use a level of indirection, using a composition of two maps to represent a single map, where the intermediate value is an attribute value. Thus, in the color example, the status map is represented by the composition of the color attribute and a "legend" map

that maps each color into a status value. This example is expanded in Section V along with examples of an off-page pointer and plot chart. The latter exploits the two spatial dimensions of a display to represent a map of two arguments.

Interference and Ambiguity

Consider the following simple display.

a	1
b	1
b	2

This display has two interpretations in the visual language:

contain({join(*a*, *1*), join(*b*, *1*), join(*b*, *2*)}) and
contain({join(*a*, *b*, *b*), join(*1*, *1*, *2*)}).

If the first interpretation is the intended option, there are numerous ways to disallow the second interpretation. Borders can be placed around the pairs:

a	1
b	1
b	2

or lines between them:

a	1
b	1
b	2

For the remainder of this paper, such distinctions are added to the examples as necessary to avoid misinterpretation at the visual-language level.

V. DERIVATIONS

Composite Sequences

Sequences can be decomposed into a sequence of sequences. This is used, for example, in line and page breaking. Consider a paragraph, **P**, as a sequence of characters. This paragraph may be broken into a sequence of lines, **BP**; each line is a sequence of characters:

P = reduce(concat, BP).

Table Derivations

Labeled Column

Consider the relation with domain and range presented in Section II:

S = {<"a","1">, <"b","1">, <"b","2">}.

Labels for the elements are introduced using a labeling map:

```
label = {| "a" -> "letter", "b" -> "letter",
           "1" -> "number", "2" -> "number" |}.
```

First **S** is represented, then the representation of **label** is added to it. **S** may be represented visually by using adjacency to represent the pair and containment to represent the set:

contain({join(*a*, *1*), join(*b*, *1*), join(*b*, *2*)}).

Rendering **join** horizontally and **contain** vertically gives the following table:

a	1
b	1
b	2

The **label** could be converted to a set of pairs and presented as a similar table. The derivation of the standard labeled-column table, however, is achieved by first inverting the **label** map using the following equivalence:

y = label(x) <=> x in label-inv(y).

The resulting **label-inv** is

label-inv = {| "letter" -> {"a","b"}, "number" -> {"1","2"} |}.

In the table generated for S, the sets **{"a","b"}** and **{"1","2"}** are represented incidentally as vertical regions; therefore, they can be used to represent **label-inv**. First, **label-inv** is converted to a set of pairs using the representation equivalence:

z = label-inv(y) <=> <y, z> in label-inv-s

giving:

label-inv-s = {<"letter", {"a","b"}>, <"number", {"1","2"}>}.

A visual language representation of this is:

contain({join(*letter*, contain({*a*, *b*}),
join(*number*, contain({*1*, *2*})})).

With the **joins** represented vertically, the resulting table is:

letter	number
a	1
b	1
b	2

Repetition of domain elements (in this case *b*) can be avoided by using the following projection equivalence:

<x, y> in S <=> ex(z) (<x, z> in S' & y in z).

This allows many possible definitions of S'; the simplest is:

S' = {<"a", {"1"}>, <"b", {"1","2"}>}.

Using the same visual representations as before and using containment to represent the new sets, the visual expression is:

contain({join(a, contain({1})), join(b, contain({1, 2})})).

Presenting the new **contains** vertically creates the following table.

letter	number
a	1
b	1
	2

Multi-Page Table

The previous derivation can be extended to handle a multi-page table. Assume that the relation is extended as follows and only four lines are allowed on a page.

```
S = {<"a","1">, <"b","1">, <"b","2">,
     <"c","2">, <"c", 3">, <"d","1">}
label = {| "a" -> "letter", "b" -> "letter",
           "c" -> "letter", "d" -> "letter",
           "1" -> "number", "2" -> "number", "3" -> "number" |}.
```

The relation is too large to fit on one page; therefore, it is split using the following representation equivalence:

<x,y> in S <=> ex(z) (z in SS & <x,y> in z).

This justifies an arbitrary split. Let us choose the following:

```
SS = {{<"a","1">, <"b","1">, <"b","2">},
      {<"c","2">, <"c","3">, <"d","1">}}
```

Each element may be represented as a table:

a	1
b	1
b	2

and

c	2
c	3
d	1

These representations have incidental representations for **{"a","b"}**, **{"c","d"}**, **{"1","2"}**, and **{"1","2","3"}**; these will be exploited in representing **label.**

Inverting **label** as before gives:

```
label-inv = {| "letter" -> {"a","b","c","d"},
               "number" -> {"1","2","3"} |}.
```

However, to exploit the available sets from the representation of SS, a representation equivalence is used that allows splitting the range elements of **label-inv** into subsets:

y in label-inv(x) <=> ex(z) (<x, z> in label-inv-s & y in z).

Choosing the subsets used to represent SS results in:

```
label-inv-s = {<"letter", {"a","b"}>, <"letter", {"c","d"}>,
               <"number", {"1","2",} >,
               <"number", {"1","2","3"} >}.
```

Using the same representation strategy as before, the following two-page table results.

letter	number
a	1
b	1
b	2

and

letter	number
c	2
c	3
d	1

Distributed Labels

Consider an alternative visual representation of a relation with labeled elements in which the labels are displayed next to the elements (i.e., the label map is distributed over the elements). The specification is:

```
S = {<"a","1">, <"b","1">, <"b","2">}
label = {| "a" -> "letter", "b" -> "letter",
           "1" -> "number", "2" -> "number" |}.
```

A map may be distributed over its domain elements by replacing each occurrence of a domain element by a pair of the element and the value of the map applied to the element. In this case, the transformation to S is given by:

```
<x, y> in S <=> <<label(x), x>, <label(y), y>> in S'.
```

This has the precondition that **label** is defined for each **x** and **y**--in this case that **domain(label)** $\supseteq$ **domain(S)** $\cup$ **range(S)**. The distribution of **label** is complete if **label(x)** appears for every **x** in the domain of **label**, which is true in this case when **domain(S)** $\cup$ **range(S)** $\supseteq$ **domain(label)**. If the distribution of **label** is complete, then the distribution serves as a complete representation of **label**. The combination of these two preconditions is:

domain(label) = domain(S) $\cup$ range(S).

This is true, so the new specification is:

S' = {<<"letter", "a">, <"number", "1">>,
<<"letter", "b">, <"number", "1">>,
<<"letter", "b">, <"number", "2">>}.

Using horizontal adjacency to represent the pairs and vertical containment to represent the set, the following table results.

letter	a	number	1
letter	b	number	1
letter	b	number	2

Diagrams and Trees

Arc Table

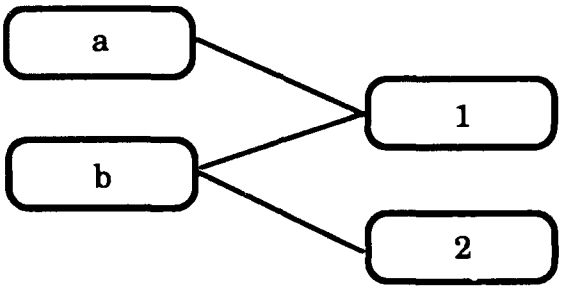
The pairs in the above examples can be represented using arcs instead of adjacency. The relation:

S = {<"a","1">, <"b","1">, <"b","2">}

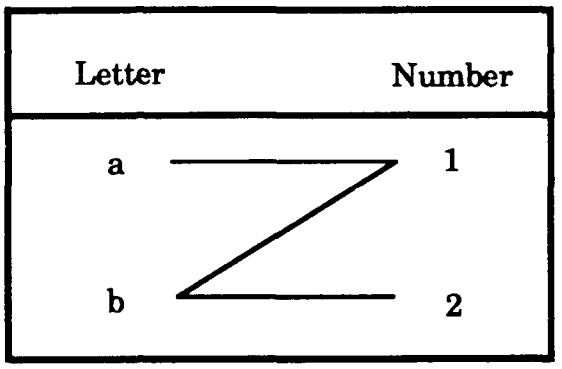
can be represented by the visual language expression:

contain({arc(*a*, *1*), arc(*b*, *1*), arc(*b*, *2*)})

giving the following diagram.



Labels can be added as before to give the following arc table.



Block Diagram

This section addresses the derivation of a diagram representing connectivity among components of a radar control system. The following abbreviation map is used to save space.

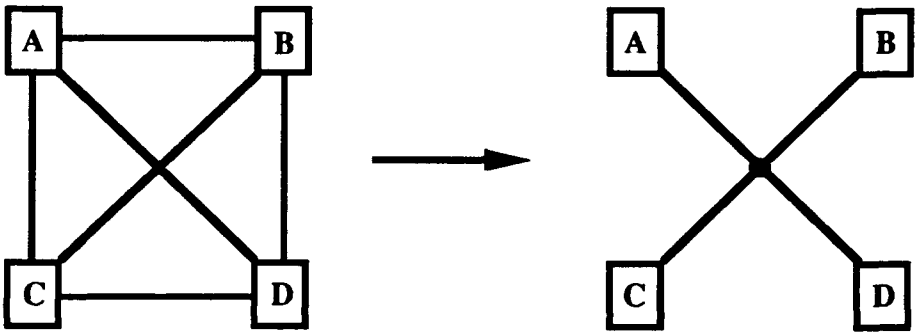
```
{| rc -> "Radar Control Panel',   fcc -> "FCC",
   tg -> "Throttle Grip",        re -> "REO",
   ss -> "Side Stick Controller", co -> "Cool",
   cw -> "Control Wiring",       fcr -> "FCR" |}
```

The connectivity specification is given by S_0, a reflexive, binary relation.

$$S_0 = \{\langle fcr, w\rangle, \langle w, fcr\rangle,$$
$$\langle fcc, re\rangle, \langle re, fcc\rangle, \langle fcc, co\rangle, \langle co, fcc\rangle,$$
$$\langle fcc, cw\rangle, \langle cw, fcc\rangle,$$

<re, co>, <co, re>, <re, cw>, <cw, re>,
<co, cw>, <cw, co>,
<rc, ss>, <ss, rc>, <rc, tg>, <tg, rc>, <rc, cw>, <cw, rc>,
<ss, tg>, <tg, ss>, <ss, cw>, <cw, ss>, <tg, cw>, <cw, tg>}

This graph has three cliques (maximal subsets of nodes where each node is connected to every other node): **{fcr, cw}**, **{fcc, re, co, cw}**, and **{rc, ss, tg, cw}**. This property can be exploited to reduce the number of arcs in a visual display by introducing a nexus point for each clique with more than two elements (i.e., by linking each node to the nexus point only rather than to all other nodes in the clique). This is illustrated in the following diagram; instead of six arcs, there are four short arcs to the nexus.



This transformation is performed in two steps: the first converts a reflexive relation to a set of cliques; the second introduces a nexus node for each clique. The first representation equivalence for the first step of the transformation is:

<x, y> in S_0 <=> ex(z) (z in S_1 & x in z & y in z & x ~= y).

This merely specifies that S_1 contains strongly connected subsets. In practice, S_1 should consist of maximal strongly connected subsets. In this case:

S_1 = {{fcr, cw}, {fcc, re, co, cw}, {rc, ss, tg, cw}}.

The second transformation step introduces the set of nexus points, N; one for each set in S_1 whose size is greater than 2.

$$x \text{ in } S_1 \iff \text{size}(x) = 2 \;\&\; \text{ex}(y, z)\ (x = \{y, z\} \;\&\; \langle y, z\rangle \text{ in } S_2)$$
$$\text{or size}(x) > 2 \;\&\; \text{ex}(n)(n \text{ in } N \;\&\; x = \{y \mid \langle n,x\rangle \text{ in } S_2\})$$

Let N = {n1, n2}, then:

$$S_2 = \{\langle fcr, cw\rangle, \langle n1, fcc\rangle, \langle n1, re\rangle, \langle n1, co\rangle, \langle n1, cw\rangle,$$
$$\langle n2, rc\rangle, \langle n2, ss\rangle, \langle n2, tg\rangle, \langle n2, cw\rangle\}.$$

Because links are used to represent the pairs, there is considerable freedom in laying out the nodes. Here connectivity was used to provide extra grouping. This provides redundancy in the display (connectivity is already represented by the links) which makes it easier for the user to see the connectivity. The transformation used exploits repetitions in the elements of S_1 (in this case cw occurs in all three elements).

$$x \text{ in } S_1 \iff \text{ex}(y, z)\ (\langle y, z\rangle \text{ in } S_3 \;\&\; x = y \cup z)$$

The equivalence justifies any splitting of the sets in S_1, but it is used to factor out a common subset; in this case {cw}:

$$S_3 = \{\langle\{cw\}, \{fcr\}\rangle, \langle\{cw\}, \{fcc, re, co\}\rangle,$$
$$\langle\{cw\}, \{rc, ss, tg\}\rangle\}.$$

The layout is a two-level process: the sets are laid out, then the elements are laid out within the sets. This structuring reduces the combinatorics of the layout algorithm and produces a layout that reflects more of the structure of the original relation. Figure 5 shows a display that satisfies the above specification, with the non-singleton sets surrounded by dashed lines (these do not appear in the actual presentation).

Containment Tree

If a map forms a tree, the map function can be shown visually by a "containment" relation such as nested boxes, nested parentheses, etc. The key is that the contained items cannot be both inside and outside the same object (this would be required if the graph were cyclical).

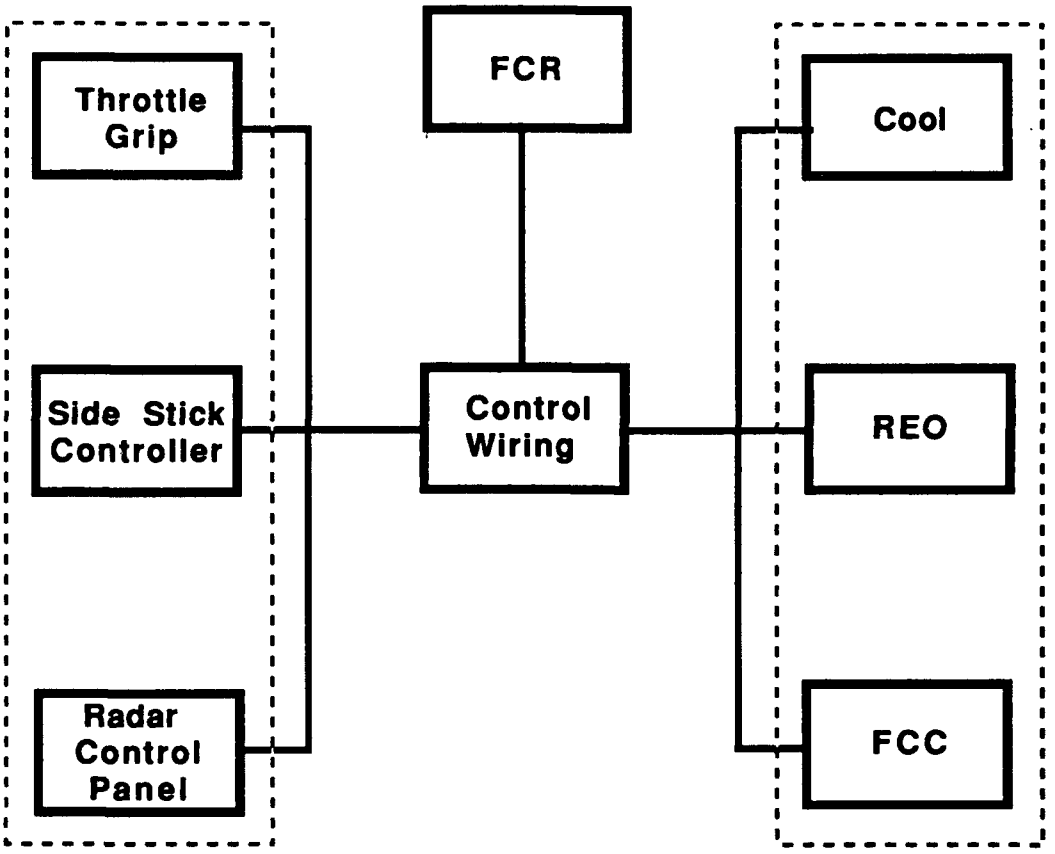


Figure 5. Sample Set Layout.

Map Decomposition

Many properties can be represented by the composition of two maps; for example, color codings, an off-page pointer, and a plot chart. A map, **h**, can be represented by two other maps, **f** and **g**, such that applying first **g**, then **f**, is equivalent to **h**; that is:

$$\mathbf{h}(x) = \mathbf{f}(\mathbf{g}(x)).$$

Color

As an example, color codings can be represented as a composition of two maps. Suppose the repair status of a component, such as **"repaired"** or **"faulty,"** is to be communicated via color coding. A component's status can be modeled by the function **status**, which maps a component into either **"repaired"** or **"faulty."**

status = {| "FCR" -> "repaired", "Cool" -> "faulty" |}

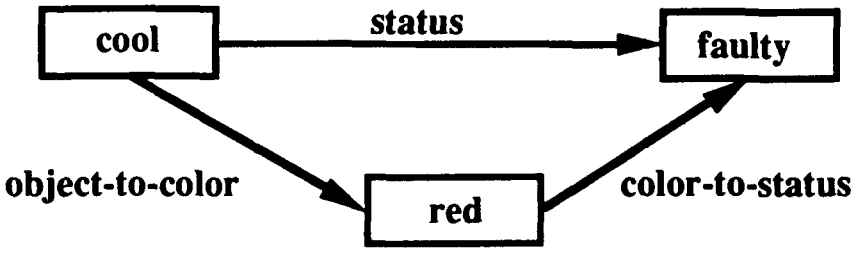
To communicate this status map, the computer will color faulty objects red and repaired objects blue. In addition, the legend or color-decoding map must be communicated to the viewer to explain what each color indicates. More formally, the function **status** can then be stated as the composition of two maps: **object-to-color** and **color-to-status**:

status(x) = color-to-status(object-to-color(x))

where object-to-color maps a faulty object to red and a repaired object to blue.

object-to-color = {| "FCR" -> blue, "Cool" -> red |}
color-to-status = {| red -> "faulty", blue -> "repaired" |}

Shown as a diagram, this example is:



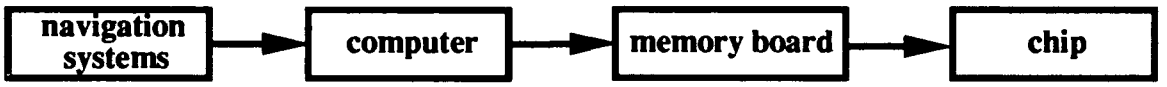
The user can check the status of the board by using the **object-to-color** map. The user must note the color of the board (in this example, red), then input that color to the **color-to-status** table to obtain the status.

Thus far, this paper has described how the user interprets the maps on the screen; now, consider how the maps are computed. First, a one-to-one correspondence is made between status values and colors, giving the map the **color-to-status** introduced above and its inverse **status-to-color**. The actual correspondence selected is not important for our purposes. Given this inverse, **object-to-color** can be computed using the formula:

object-to-color(x) = status-to-color(status(x)).

Off-Page Pointers

In this section, off-page pointers are derived as another example of using map composition to communicate graphically. Suppose a system-subsystem relation is represented visually as links between the visual representation of systems and subsystems.



Each link is called a system-to-subsystem link. Now, assume there is insufficient space to show all four levels of this tree. One solution is to use off-page pointers. Thus, the first part of the tree would appear on the first page:

and the remainder would appear on page 3:

An off-page or out-of-window pointer is formalized as a map decomposition. The composition of two maps to represent **system-to-subsystem** may be defined as:

system-to-subsystem(x)
= label-to-subsystem(system-to-label(x))

where the map **system-to-label** maps a system to a label, and the map **label-to-subsystem** maps the label to a subsystem. Then, an implementation must be selected for the two maps. In the example above, the same visual representation was used (directed arcs) to represent the two composed maps as was used to represent the original map.

Plot Chart

A plot chart exploits the two spatial dimensions of visual space to display a two argument map: **P X Q -> R**. It is an example of map decomposition extended to two dimensions. **P** is

mapped to one dimension--say the x-dimension--and Q is mapped to the other--the y-dimension. Then, for each pair **<a, b>** in the domain of the map (**a in P** and **b in Q**), the corresponding map value is displayed at the position where the x-value is **a** and the y-value is **b**.

As an example, a plot chart display of the following set of pairs was derived. This is considered a characteristic map.

S = {<"a","1">, <"b","1">, <"b","2">,
<"c","2">, <"c","3">, <"d","1">}

The first step is to represent S as its characteristic map S_m:

x in S <=> S_m(x).

Thus:

S_m(<"a","1">) = true, S_m(<"b","1">) = true, S_m(<"b","2">) = true,
S_m(<"c","2">) = true, S_m(<"c","3">) = true, S_m(<"d","1">) = true

and S_m is **false** elsewhere.

Next, S_m was decomposed into two maps as in the previous sections:

S_m(<x, y>) = position-to-Boolean(pair-to-position(<x, y>))].

The key to plot chart derivation is to choose **pair-to-position** to be the pairing of the **x-position** of the first argument and the **y-position** of the second argument:

pair-to-position(<x, y>) = <x-position(x), y-position(y)>.

Then, if the following relations are selected:

x-position("a") = 1, x-position("b") = 2, x-position("c") = 3,
x-position("d") = 4
y-position("1") = 2, y-position("2") = 1, y-position("3") = 0

it follows from the decomposition equation for S_m that:

position-to-Boolean(<1, 2>) = true,
position-to-Boolean(<2, 2>) = true,
position-to-Boolean(<2, 1>) = true,
position-to-Boolean(<3, 1>) = true,
position-to-Boolean(<3, 0>) = true,
position-to-Boolean(<4, 2>) = true

and **position-to-Boolean** is **false** elsewhere.

There is still freedom in choosing the **y-position** of **"a"**, **"b"**, **"c"** and **"d"**; and the **x-position** of **"1"**, **"2"** and **"3"**. Choosing a single **y-position** and a single **x-position** has the desirable side effect of representing the sets **{"a", "b", "c", "d"}** and **{"1", "2", "3"}**. Choosing this single **y-position** to be 3 and this single **x-position** to be 1, and representing **true** by *x* and **false** by an empty region, results in the following plot chart, where the bottom-left corner has position **<0, 0>**.

	a	b	c	d
1	x	x		x
2		x	x	
3			x	

VI. CONCLUSIONS

We have developed a formal framework for transforming objects into different representations. The framework allows for a fine-grained control for the synthesis of graphics. It also provides handles for the inclusion of human factors and interactive displays. The framework employs a data language and a visual representation relation. The framework allows the formalization of conditions for determining visual interference or visual ambiguity.

A major issue that was resolved is that the theory does scale up to multi-page diagrams. The rules that were presented allow objects to be broken into sub-objects (to appear on separate pages); distributed maps allow the sub-objects to retain their proper labeling.

This paper demonstrated how composite maps can model visual techniques, such as color coding and off-page pointers, and described their use in deriving plot charts.

The refined theory of adjacency and connectedness clarifies and illustrates the inter-relationships of tables and diagrams. This theory will also lead to a more seamless integration of table-like and diagram-like entities. The table derivations also allow better handles on rows, columns, and headings.

Our formalization provides a deterministic procedure for interpreting displays relationally, but a highly non-deterministic procedure for generating displays from relational data. Methods for controlling this non-determinism must be examined. In particular, a criterion needs to be incorporated for determining which displays are good, such as a cost function.

REFERENCES

Bertin, J. (1983). Semiology of graphics. Milwaukee, WS: University of Wisconsin Press.

MacKinlay, J. (1986). Automating the design of graphical presentations of relational information. ACM Transactions on Graphics, 5 (4), 110-141.

Westfold, S. J., Green, C. C., & Zimmerman, D. L. (1990). Automated design of displays for technical data (AFHRL-TP-90-66). Wright-Patterson AFB, OH: Logistics and Human Factors Division, Air Force Human Resources Laboratory.